Tomáš Meravý Murárik
ais:127232

Časť 1:

Popis projektu

Cieľom projektu bolo vytvoriť neurónovú sieť schopnú predikovať strednú hodnotu ceny domu na základe dostupných údajov o okresoch v Kalifornii. Mnou použita dátová množina obsahovala 20 640 prípadov a 8 vstupných atribútov, vrátane stredného príjmu, veku nehnuteľnosti, priemernej obsadenosti a geografických údajov.

Architektúra modelu(class HousingModel)

Použitá bola neurónová sieť s nasledujúcou štruktúrou:

1. **Vstupná vrstva:** 8 neurónov (počet vstupných atribútov po predspracovaní).
2. **Skryté vrstvy:**
    1. vrstva: 64 neurónov, aktivačná funkcia ReLU.
    2. vrstva: 128 neurónov, aktivačná funkcia ReLU.
    3. vrstva: 64 neurónov, aktivačná funkcia ReLU.
    4. vrstva: 32 neurónov, aktivačná funkcia ReLU.
3. **Výstupná vrstva:** 1 neurón (predikovaná hodnota ceny domu).

Použité hyperparametre

Parameter	Hodnota
Počet epoch	100
Veľkosť dávky (batch)	64
Optimalizátory	SGD, SGD + moment, ADAM
Rýchlosť učenia	0.001
Aktivačná funkcia	ReLU
Strata (Loss function)	MSELoss

Počas celého projektu som využíval nielen pytorch a numpy ale aj pandas a sklearn knižnice. Tie mi dovolili pomocou predrobených funkcií ľahko predspracovať dáta.

Predspracovanie údajov

1. **Čistenie dát:** Čistenie dát bolo urobené pomocou funkcie .fillna z pandas knižnice a hodnota použitá bola median.

Tomáš Meravý Murárik
ais:127232

2. **Konverzia kategórií:** Atribút `ocean_proximity` bol transformovaný na numerické hodnoty tým , že bol jeho tip zmenený na ‘category’ a potom každá kategória bola zmenená za jej číslo.
3. **Normalizácia:**
    - Vstupné atribúty boli normalizované pomocou `StandardScaler` z knižnice Scikit-learn.
    - Cieľové hodnoty (`median_house_value`) boli škálované, aby tréning bol stabilnejší.
4. **Rozdelenie dát:**Rozdelenie dát bolo urobené pomocou sklearn funkcie train_test_split v pomere 0.2
5. **HousingDataset:**Následne je nutné zmeniť dáta na formát ktorému rozumie DataLoader a to môžme docieliť classou ktorá bude mať definované správne funkcie __getitem__ a __len__.

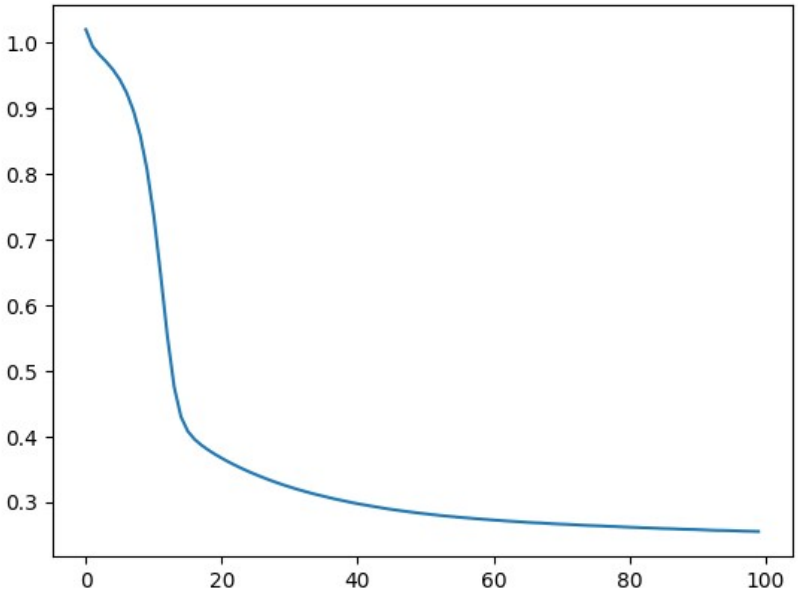
Porovnanie optimalizačných algoritmov

1. **SGD**
    - Trénovacia chyba (MSE) po 100 epochách: **0.2549**
    - Pomalejší konvergencia, vyžaduje viac epoch pre dosiahnutie presnejších predikcií.
2. **SGD s momentom**
    - Trénovacia chyba (MSE) po 100 epochách: **0.1803**
    - Rýchlejšia konvergencia v porovnaní s obyčajným SGD.
3. **ADAM**
    - Trénovacia chyba (MSE) po 100 epochách: **0.112**
    - Najrýchlejšia konvergencia a najnižšia chyba. Tento algoritmus sa ukázal ako najvhodnejší pre tento problém.

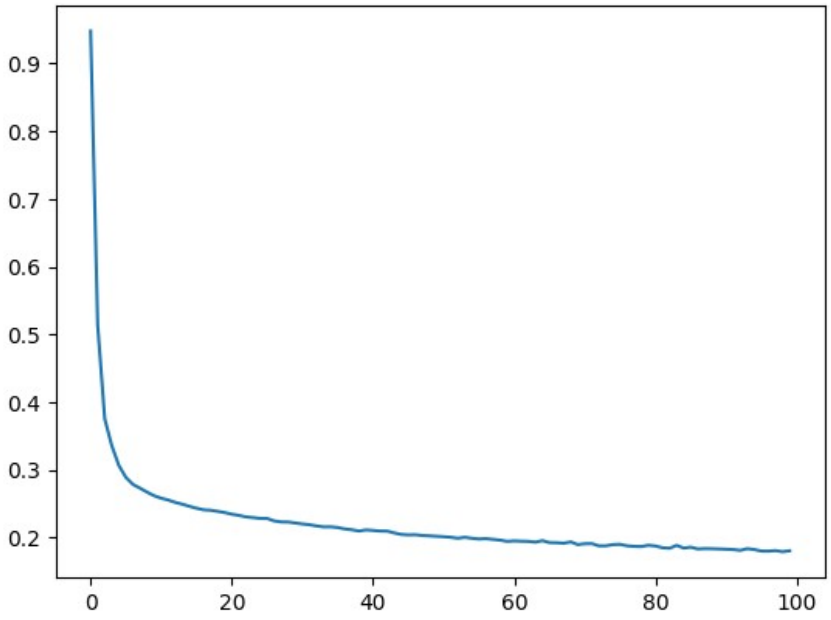
Tomáš Meravý Murárik
ais:127232

Grafy priebehu tréningu

1. **SGD**

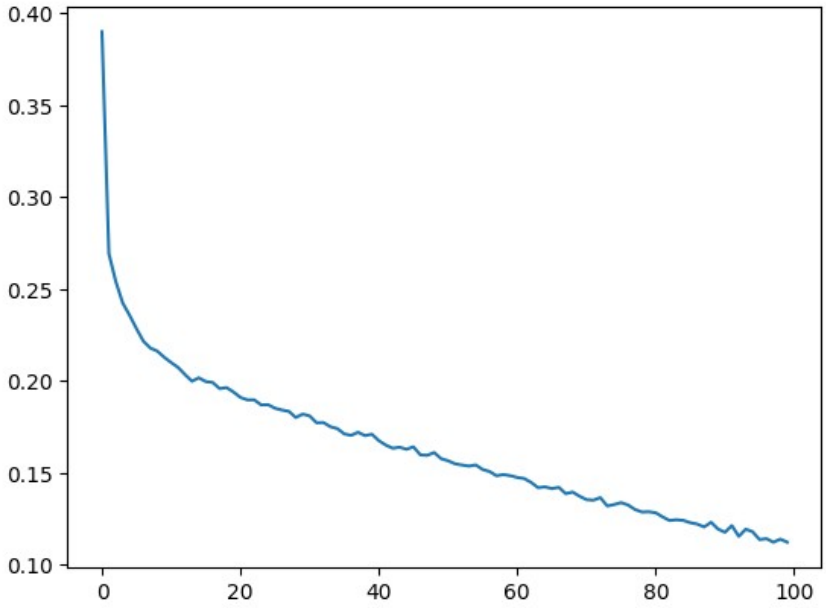


2. **SGD s momentom**



Tomáš Meravý Murárik
ais:127232

3. **ADAM**



Zhodnotenie výsledkov

Na základe výsledkov experimentu som usúdil, že optimalizačný algoritmus ADAM mal najlepšie výsledky, pokiaľ ide o rýchlosť konvergencie a dosiahnutú presnosť. SGD s momentom bol taktiež úspešný, no vyžadoval o niečo viac času na dosiahnutie porovnateľnej chyby. Normálny SGD mal najpomalší priebeh tréningu, čo naznačuje, že bez ďalších úprav je menej vhodný na tento typ problému.

Celkovo bolo možné dosiahnuť dobré predikcie cien nehnuteľností pomocou jednoduchej doprednej neurónovej siete, pričom predspracovanie dát a výber optimalizačného algoritmu boli veľmi dôležité.

Tomáš Meravý Murárik
ais:127232

Časť 2:

Prehľad:

V tomto projekt implementujem plne funkčný algoritmus spätného šírenia (backpropagation) na trénovanie neurónovej siete na riešenie problému XOR. Sieť je navrhnutá s modulárnou architektúrou, kde je každá vrstva, aktivačná funkcia a chybová funkcia implementovaná ako samostatná trieda. Trénovanie sa vykonáva pomocou algoritmu gradientného zostupu, pričom je implementovaná verzia s aj bez momenta.

Architektúra siete:

- **Vstupná vrstva**: 2 neuróny predstavujúce 2 vstupy pre problém XOR teda jednu z možností [0, 0], [0, 1], [1, 0], [1, 1].
- **Skrytá vrstva**: 4 neuróny s aktivačnou funkciou Tanh/ReLU/Sigmoid.
- **Výstupná vrstva**: 1 neurón s aktivačnou funkciou Tanh (predstavujúci výstup XOR).
- **Chybová funkcia**: Chybová funkcia :MSE.
- **Optimalizátor**: Gradientný zostup s voliteľným momentom.

Optimalizačná funkcia Tanh

```
class Tanh:
    def forward(self, x):
        self.output = np.tanh(x)
        return self.output
    def backward(self, grad_output):
        return grad_output * (1 - self.output**2)
```

Kroky:

1. **Dopredný smer**:

Tomáš Meravý Murárik
ais:127232

    - Vstupné dáta sa najprv prechádzajú cez prvú lineárnu vrstvu (`LinearLayer/LinearLayerWithMedian`), následne cez aktivačnú funkciu Tanh (`Tanh`).
    - Výstup zo skrytej vrstvy sa ďalej posiela cez druhú lineárnu vrstvu 8a nakoniec prechádza cez ďalšiu aktivačnú funkciu Tanh.
2. **Spätný smer**:
    - Vypočíta sa gradient chyby vzhľadom na výstupnú vrstvu a chyba sa propaguje späť cez sieť.
    - Vypočítajú sa gradienty pre váhy a biasy v oboch vrstvách pomocou reťazového pravidla.
    - Váhy a biasy sa aktualizujú pomocou vypočítaných gradientov a učebnej rýchlosti.
3. **Aktualizácia parametrov**:
    - Váhy a biasy sa aktualizujú pomocou gradientného zostupu. Momentum je aplikované pridaním rýchlostného členu, ktorý vyhladí aktualizácie.

Kľúčové komponenty:

- **Trieda LinearLayer**: Implementuje základnú pripojenú vrstvu so spätným smerom. Podporuje aktualizácie váh.
- **Trieda LinearLayerWithMomentum**: Podtrieda triedy `LinearLayer`, ktorá implementuje momentum v aktualizáciách váh. Jediný rozdiel od LinearLayer je v update funkcií.
- **Aktivačné funkcie**: Tanh, Sigmoid, ReLU (implementované ako samostatné triedy), aj keď v tejto úlohe je použitá len funkcia Tanh, funkciu môžeme ľahko zmeniť tým , že v model variable zmeníme Tanh na našu požadovanú funckiu.
- **Chybová funkcia**: Trieda MSELoss, ktorá vypočíta hodnotu chyby a jej gradient.
- **Trieda SequentialModel**: Ťahá vrstvy a aplikuje dopredný a spätný smer, ako aj aktualizácie.

Výsledky tréningu:

- Tréningová strata klesá konzistentne, čo naznačuje, že sieť sa efektívne učí.

Tomáš Meravý Murárik
ais:127232

- Straty sú printnutú každých 50 epoch, kde v priemere už pri 200 epochoch mala byť loss menšia ako 0.001
- Na konci 500 epoch dosahuje sieť nízk u hodnotu straty, čo naznačuje, že sa sieť úspešne naučila modelovať XOR problém.

Príklad úspešného zbehnutia programu:

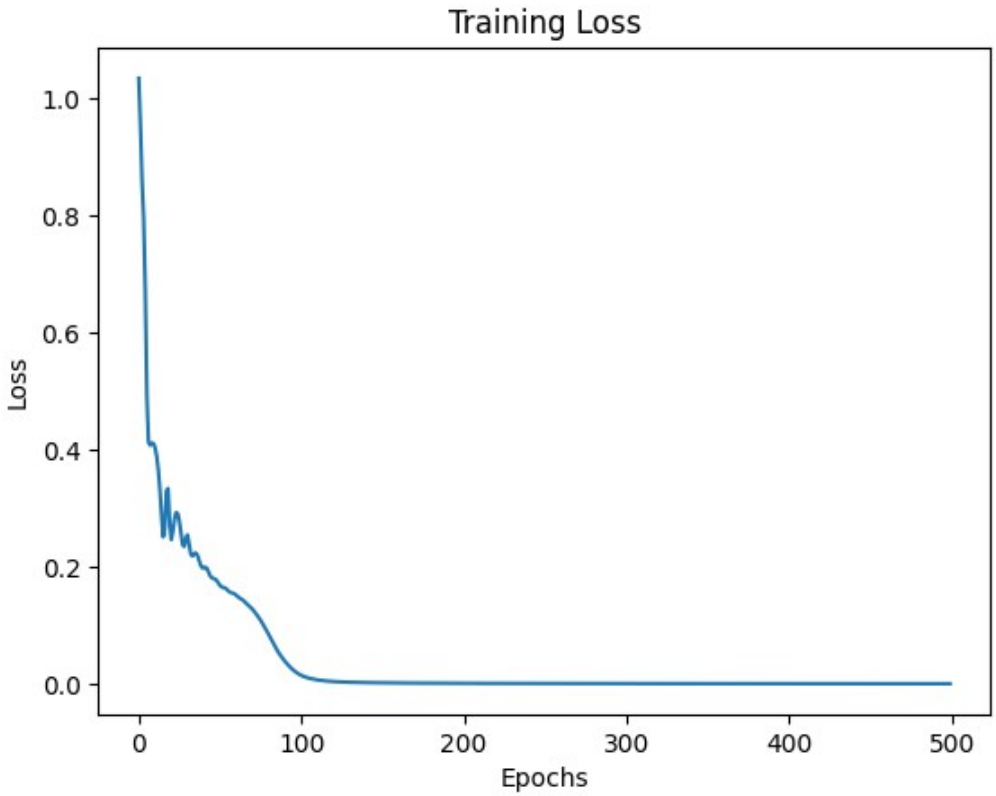
```
50  Loss: 0.17224520915035277
100 Loss: 0.01587416746000546
150 Loss: 0.0017739357019554921
200 Loss: 0.0009646397534647574
250 Loss: 0.000657975230070605
300 Loss: 0.0004945566608261768
350 Loss: 0.00039360253507392734
400 Loss: 0.0003253786041447496
450 Loss: 0.0002763707342858691
500 Loss: 0.00023956851811000 02
```

Testovacie výsledky:

Po tréningu sieť dosiahne veľmi nízku testovaciu stratu, čo znamená, že model sa dobre generalizuje na trénovacie dáta. Strata na testovacej množine je veľmi blízka nule, čo indikuje vysokú presnosť predikcie.

Training Loss LinearLayer :XOR

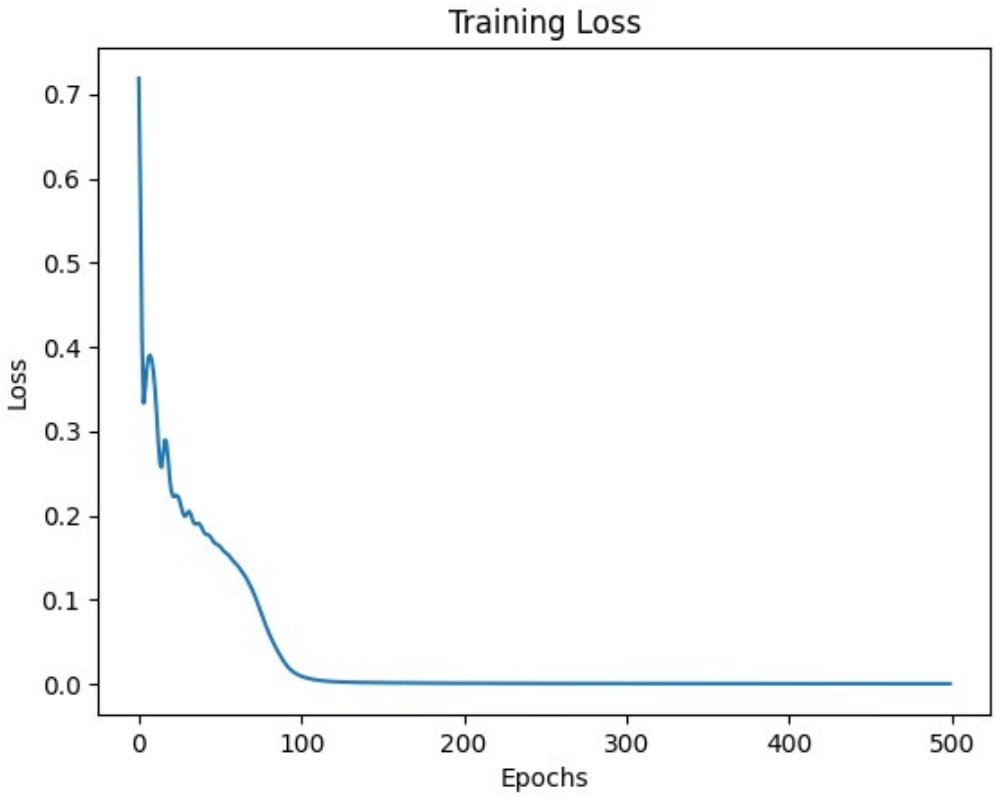
Tomáš Meravý Murárik
ais:127232



Training Loss LinearLayerWithMomentum:XOR

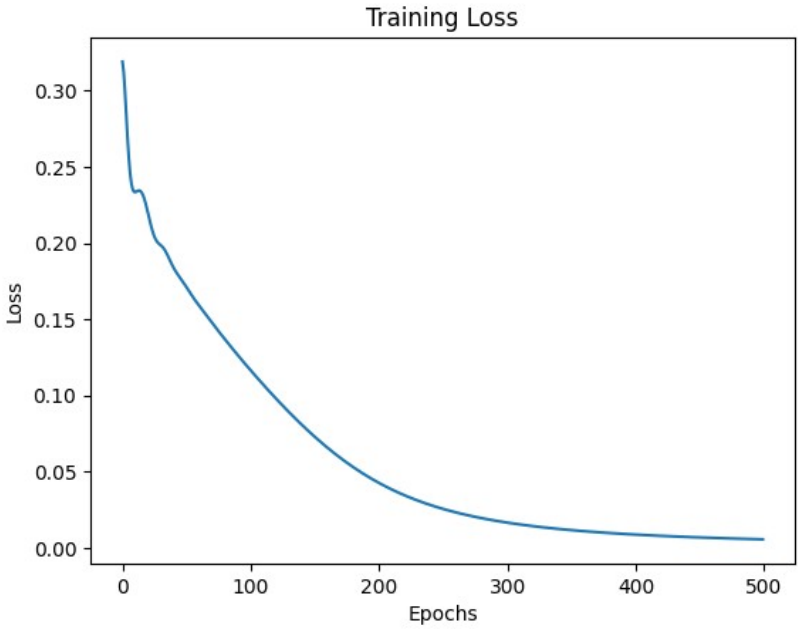
learning rate:0.1 Test Loss:0.00017913816417746786

Tomáš Meravý Murárik
ais:127232



Training Loss LinearLayerWithMomentum:XOR

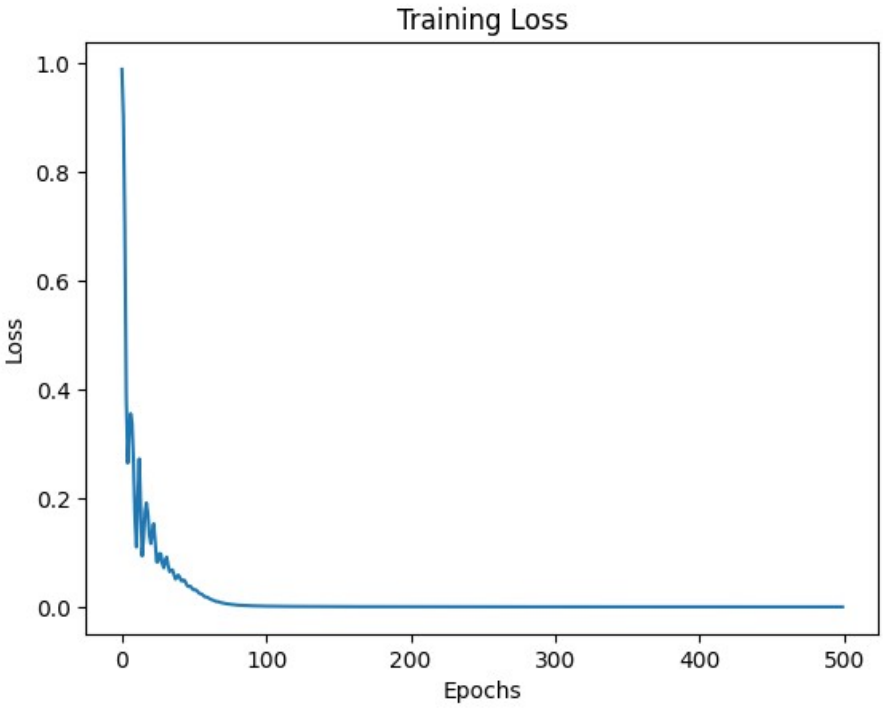
learning rate:0.01 Test Loss:0.0055282215595958695



Tomáš Meravý Murárik
ais:127232

Training Loss LinearLayerWithMomentum:AND

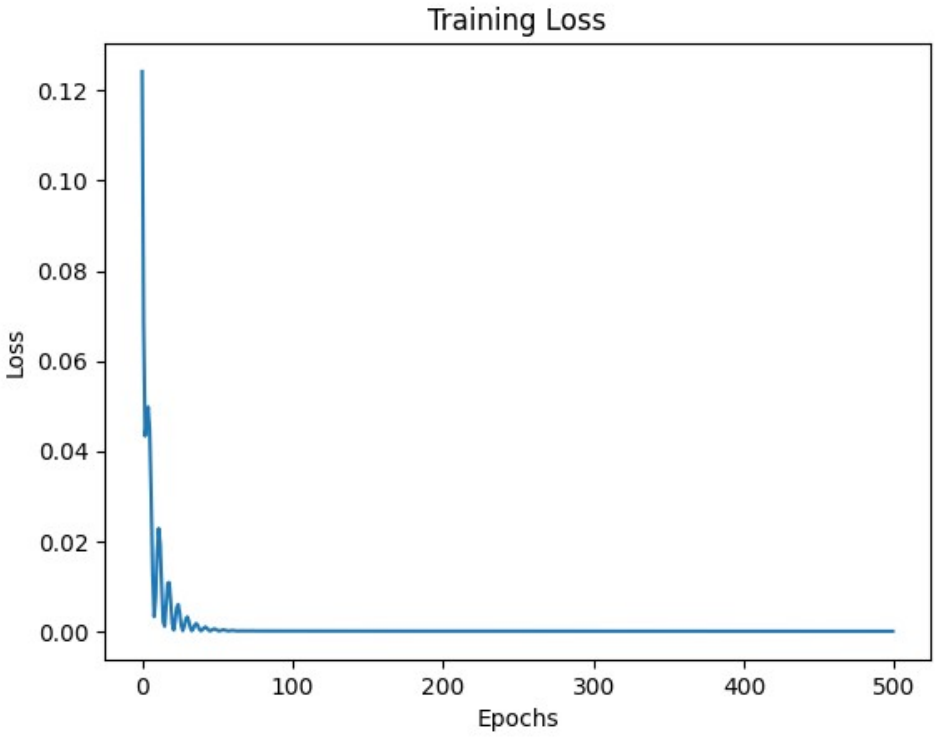
Test Loss:9.46510810227897e-05



Training Loss LinearLayerWithMomentum:OR

Test Loss:3.908560562805316e-05

Tomáš Meravý Murárik
ais:127232



Záver:

- Neurónová sieť sa dokázala efektívne naučiť riešiť problém XOR pomocou algoritmu spätného šírenia a najlepše výsledky na XOR problém mala pri learning rate 0.1 .
- Použitie momenta mi pomohlo zrýchliť konvergenciu v porovnaní s obyčajným gradientným zostupom.
- Sieť dosiahla nízku testovaciu stratu, čo znamená dobrú generalizáciu na trénovacie dáta.

Ďalšie vylepšenia:

- Experimentovanie s rôznymi aktivačnými funkciami (napr. ReLU) môže priniesť lepšie výsledky pri trénovaní na iných problémoch, ako sú AND alebo OR.
- Testovanie rôznych hodnôt učebnej rýchlosti a momenta môže optimalizovať tréningový proces a zlepšiť rýchlosť konvergencie.

Tomáš Meravý Murárik
ais:127232

Informácie som čerpal zo zdrojov :
https://github.com/aliejabbari/Optimizations-ADAM-Momentum-SGD/blob/main/neural_network_optimization.ipynb
https://www.geeksforgeeks.org/how-to-implement-neural-networks-in-pytorch/